[19] STATE INTELLECTUAL PROPERTY OFFICE OF PRC

[51] Int. Cl[7]
G06F 9/38

[12] SPECIFICATION OF PATENT APPLICATION FOR INVENTION

[21] Application No.: 01,131,569.5

[43]Publication Date: May 15, 2002

[11] Publication Number: CN 1,349,160A

[22] Filing Date: November 28, 2001

[21] Application No.: 01,131,569.5

[71] Applicant: National University of Defense Technology of the PLA

Address: Computer College of National University of Defense Technology of the PLA
47 Yanwachi Center Street, Changsha, Hunan Province
410073
PR China

[72] Inventors: Dai Kui, Wang Zhiying, Shen Li, Wang Ronghui, Wang Lei, Zhang Chunyuan, Wang Mingshi, Liu Fang

[74] Patent Agency: Hunan Zhaohong Patent Law Office

Agent: Zhao Hong

Including 3 pages of Claims, 10 pages of Description and 2 pages of Accompanying Drawings

---

[54] Method for Eliminating Pipeline Control Related Delay

[57] Abstract:

The invention discloses a method for eliminating the pipeline control related delay, for efficiently eliminating the pipeline control related delay so as to increase the performance of microprocessor with simpler hardware and lower power consumption. The technical solution is as follows. A compiler determines all possible transferring target addresses for a branch type instruction and inserts a prefetch instruction. All subsequent instructions of the present instruction are read by two instruction fetch components in advance, but only the instructions provided by one of the two instruction fetch components are selected by a selector based on the decoding results of the present branch type instruction for decoding and executing. There are three prefetch instructions, namely, fetch addr1 & addr2, fetch addr, fetch stack, corresponding to various branch type instructions and executed by the instruction fetch component. The instruction prefetch is performed in basic blocks. The invention has advantages of lower power consumption and complexity of hardware, high ratio for eliminating control related delay, high performance and cost ratio and reduced useless prefetched instructions.

Method for Eliminating Pipeline Control Related Delay

Field of the Invention

The present invention generally relates to a method for eliminating the pipeline control related delay in the microprocessor design, particularly in the design of an embedded microprocessor with low power consumption and complexity of hardware.

Description of the Related Art

At present, pipeline control related delay in the microprocessor design is eliminated mainly in two ways, i.e., branch prediction and delayed branch. The locality principle in program running is utilized in the branch prediction method to predict whether the next branch transfer will succeed or not based on the statistic information of the execution results of the branch type instruction. The effect of the branch prediction depends upon its accuracy, and furthermore, is closely related to the overhead caused by the branch prediction. The pipeline control related delay depends upon the configuration of the pipeline, the prediction method and the restoration policy adopted after a prediction error occurs. This method has a disadvantage in that it demands great hardware support, such as prediction table and restoration components after a prediction error occurs, resulting in high overhead and power consumption. The basic principle of the delayed branch method lies in that instructions irrelated to the branch type instruction are executed during the control related pause cycle so as to cover the cycle. This method also has a disadvantage that it is impossible to fill all the branch delay slots and to ensure that all instructions called are to be executed. In case any instructions that are not necessarily executed are called, improved performance is not provided in a real sense.

Embedded microprocessors, which are mainly deployed in areas such as domestic appliances, cell phones, microcontrollers, and the like, demand low power consumption, and none of the above methods is a satisfactory solution as they cannot substantially improve its performance either because they fail to meet the requirement for low power consumption and complexity of hardware or because they are not efficient enough in eliminating the control related delay. These methods cannot efficiently eliminate the control related delay in the pipeline even in the general purpose microprocessor design.

Summary of the Invention

The technical problem the present invention is to solve is to efficiently eliminate the pipeline control related delay in the embedded microprocessor with low complexity of hardware and power consumption, to thereby improve the performance of the microprocessor.

The technique scheme is as follows. A compiler in the compiling module determines all possible transferring target addresses for a branch type instruction and inserts a

prefetch instruction. Two instruction fetch components are disposed in the instruction fetch module to read all subsequent instructions of the present instruction in advance. A selector is disposed in the instruction decoding and execution module to select instructions provided by one of the two instruction fetch components based on the decoding results of the present branch type instruction for decoding and execution by the instruction decoding and execution module, to thereby eliminate the pipeline control related delay. It has not been reported that the pipeline control related delay is eliminated with this method at home or abroad.

In the invention there are eight terms involved, i.e., pipeline, branch type instruction, pipeline related, control related, control related delay, instruction prefetch, prefetch instruction, basic block, which are respectively defined as:

1) Pipeline, i.e., the specific realization of the instruction execution pipelining technique which divides the instruction execution procedure of a microprocessor into a number of sub-procedures with each sub-procedure able to be effectively executed on its special functional section simultaneously with other sub-procedures. The number of the stages that a pipeline can be divided into varies in different microprocessor designs. A pipeline generally includes five stages: instruction fetch, decode, execution, memory access, and write-back.

2) Branch type instruction: generally referring to all the instructions that can change the value of a program counter. It falls into four categories: conditional transfer instruction, direct unconditional transfer instruction, indirect unconditional transfer instruction, and procedure return instruction such as a return statement.

3) Pipeline related: referring to pipeline pause caused by the interdependency between instructions.

4) Control related: referring to the pipeline related that is caused by a branch type instruction.

5) Control related delay: referring to number of the pipeline pause clock cycle that results from control related.

6) Instruction prefetch: referring to the operation that an instruction is fetched from a memory in advance prior to being executed.

7) Prefetch instruction: referring to an instruction for performing prefetch.

8) Basic block: a basic unit of a program with only one entry, i.e., the first statement of the basic block, and only one exit, i.e., the last statement of the basic block. A program can always be divided into a number of basic blocks, each of which, in turn, includes at least two instructions with the one at the exit being branch type.

The execution procedure is as follows:

1. A compiler in the compiling module determines all possible transferring target addresses for a branch type instruction and inserts a prefetch instruction.
2. When the program begins running, one of the two instruction fetch components reads in the first basic block while the other stays idle.

For each basic block in the program,

(a) The instruction fetch component that reads in the basic block sequentially reads in each instruction in the basic block and determines whether it is a prefetch instruction. The fetch component sends it to the other instruction fetch component for execution if the presently read-in instruction is a prefetch instruction, or otherwise the fetch component sends the instruction to the decoding component in the instruction decoding and execution module.

(b) After the last instruction of the basic block, i.e., the branch type instruction, finishes decoding, a selector in the instruction decoding and execution module selects a basic block from the instruction fetch component as a subsequent basic block based on the decoding results:

(i) Assuming that the two instruction fetch components are IF_0 and IF_1, respectively, and the instructions in IF_0 are currently being executed, then IF_1 executes the prefetch instruction and prefetches the subsequent target instruction for the present basic block. When the instructions are executed sequentially, that is, the instruction that ends decoding is not a branch type instruction or is a branch type instruction whose transferring condition is "False", the instructions in IF_0 are selected for decoding; when the instruction that ends decoding is a branch type instruction whose transferring condition is "True", the instructions in IF_1 are selected for decoding.

(ii) If at present the pipeline is executing the instructions in IF_1, then the selection policy is just the opposite, that is, if the instructions are executed sequentially according to the program, i.e., the instruction that ends decoding is not a branch type instruction or is a branch type instruction whose transferring condition is "False", the instructions in IF_1 are selected for decoding; when the instruction that ends decoding is a branch type instruction whose transferring condition is "True", the instructions in IF_0 are selected for decoding.

Accordingly, the two instruction fetch components work in parallelism with one providing instructions for the instruction execution component and the other performing instruction prefetch, so that all the possible branch target instructions have already been stored in the corresponding instruction fetch components before the decoding of the branch type instruction ends.

Compared with conventional compilers, the compiler of the invention has two additional special functions, namely, determining all the possible transferring addresses for the branch type instruction and inserting a prefetch instruction into the basic blocks according to the various categories of the branch instructions. The flow for the compiler to insert the prefetch instruction runs as follows. The compiling routine reads in turn each instruction from the program code and, when a branch type instruction appears, which indicates that the end of the present basic block is reached, inserts a corresponding prefetch instruction behind the first instruction of the basic block to which the branch type instruction belongs according to the category of the

branch type instruction.

The invention designs three prefetch instructions, i.e., fetch addr1 & addr2, fetch addr, and fetch stack, corresponding to the various branch type instructions that mainly fall into four categories.

1) Conditional branch instruction. For this instruction, the transferring may succeed or fail and there are two subsequent basic blocks, both of which should be prefetched for the present basic block. There are two possible transferring addresses, one stored in the instruction and the other being the address of the following instruction. The first prefetch instruction fetch addr1 & addr2 is now inserted behind the first instruction of the basic block to which the branch instruction belongs, to prefetch the two basic blocks that begin at the addresses addr1 and addr2. addr1 is obtained by decoding the present branch type instruction and add2 is the address of the following instruction with a value of the sum of the branch type address and instruction length (in terms of bytes).
2) Direct unconditional branch instruction, where the transferring is always successful and there is only one subsequent basic block, so the transferring target address can be obtained in compiling, and only the target basic block is to be prefetched. There is only one possible transferring address, stored in the instruction. The first prefetch instruction fetch addr is now inserted behind the first instruction of the basic block to which the branch instruction belongs, to prefetch the basic block that begins at the address addr, which is obtained from the branch type instruction decoding.
3) Indirect unconditional branch instruction, where the transferring is always successful, but the transferring target address cannot be obtained in compiling as it is stored in a register. This category of branch type instruction is not processed.
4) Procedure return statement, where the transferring is always successful and there is only one subsequent basic block. These statements often occur at the procedure calling returns. The present invention stores the procedure calling return address (i.e., the prefetch address) using a stack as nesting may occur in procedure calling. For each procedure calling, the return address is stored at the stack top location, from where the prefetch address is obtained during prefetching. The prefetch instruction fetch stack is now inserted behind the first instruction of the basic block to which the procedure return instruction belongs, to prefetch the basic block that begins at the stack top location address.

Different from other instructions, the prefetch instruction is executed by an instruction fetch component. The prefetch instruction coding may vary corresponding to various RISC (Reduced Instruction Set Computers) instruction sets, but all fall into the scope of the present invention as long as it is utilized to realize the same function as the present invention. The prefetch is done in basic blocks and all the prefetched instructions should be executed except some of the subsequent instructions that are prefetched when the conditional branch instruction transfer fails.

If all the possible target instructions have been read in before the branch type instruction decoding ends, correct subsequent instructions may be selected according to the results of the branch type instruction decoding for decoding and execution. When the present invention is realized in an FastDLX simulator (standard CPU simulator) and tested by SPECint95 benchmark set provided by the System Performance Evaluation Cooperative Consortium, the probability that the branch target instructions have already been read in when the branch type instruction decoding ends is up to 99.3% with the indirect unconditional branch instruction, which holds a very low percentage in the procedure, not taken into account, and is 93% otherwise.

The present invention has the following advantages:

1) Low complexity and power consumption of hardware realization. Compared with branch prediction technology, the present invention utilizes simple logic control components, omitting complex branch prediction hardware and the restoration hardware for prediction errors, and therefore may greatly reduce the difficulty and complexity of hardware realization.
2) High elimination rate of the control related delay. The present invention selects branch target addresses directly according to the decoding results of the branch type instruction, so the instruction prefetch may ensure that most of the instructions have already been read in before the branch instruction decoding ends, and therefore, most majority of the control related delay will be eliminated.
3) The prefetch is performed in basic blocks and the subsequent basic blocks are simultaneously read in when the present basic block is performed, therefore, the prefetch instruction is issued at an early time, which ensures adequate time for instruction prefetch. In the meanwhile, all the prefetched instructions, except those subsequent instructions that are prefetched when the conditional branch instruction transfer fails, are to be executed, which effectively reduces the amount of useless instruction prefetch.

The present invention meets the objects of effectively eliminating pipeline control related delay and improving the performance of the embedded microprocessor with low complexity of hardware and low power consumption.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a flow chart of a compiler of the present invention inserting a prefetch instruction;

Figure 2 illustrates the general building block of logic of the present invention;

Figure 3 is a space vs. time diagram of the non-branch type instruction of a conventional microprocessor in a five-stage pipeline;

Figure 4 is a space vs. time diagram of the branch type instruction of a conventional microprocessor in a five-stage pipeline;

Figure 5 is a space vs. time diagram of the branch type instruction in a five-stage pipeline after the present invention is adopted;

Figure 6 illustrates the testing results of the present invention with respect to SPECint 95 benchmark;
Figure 7 illustrates a comparison of a control related delay eliminating method with the present invention and other methods.

Detailed description of the invention

Fig 1 is a flow chart of a compiler of the present invention inserting a prefetch instruction. The compiling program sequentially reads each instruction in the program code. The appearance of a branch type instruction indicates that the end of the present basic block is reached and then the compiler inserts a corresponding prefetch instruction behind the first instruction of the basic block the branch type instruction belongs to according to the category of the branch type instruction. The compiler can always come across a branch type instruction since the program always ends with a procedure return statement.

Fig 2 illustrates the general building block of logic of the invention, which comprises of a compiling module, instruction fetch module, and instruction decoding and execution module.

The compiling module is designed mainly to determine all the possible transferring addresses for the branch type instruction and insert a prefetch instruction according to the category of the branch type instruction. The compiler sequentially reads each instruction in the source program. When a branch type instruction is read, the compiler inserts a corresponding prefetch instruction behind the first instruction of the basic block the branch type instruction belongs to. This is done as follows. For a conditional branch type instruction, since there are two subsequent basic blocks and hence two corresponding prefetch addresses, prefetch instruction fetch addr1 & addr2 is inserted with addr1 obtained by decoding the branch type instruction and addr2 being the address of the following instruction with a value of the address of the branch type instruction plus the instruction length (in terms of byte). For a direct unconditional branch type instruction, there is only one subsequent basic block and hence one corresponding prefetch address, and prefetch instruction fetch addr is inserted with addr obtained by decoding the instruction. For the procedure return statement, there is only one subsequent basic block and hence one corresponding prefetch address stored in the stack top location, and prefetch instruction fetch stack is inserted. The compiled program code is stored in a memory.

The instruction fetch n module is designed mainly to provide instruction to the instruction decoding and execution module and perform instruction prefetch, and the module functions through two instruction fetch components. The instruction fetch component IF_0 reads instructions from the instruction Cache through the port 0, while the instruction fetch component IF_1 reads instructions from the instruction Cache through the port 1. When the program begins running, it selects the instructions provided by IF_0 for decoding and execution and IF_1 stays idle. A prefetch instruction

is read in and then transferred to IF_1 by IF_0 for IF_1 to perform prefetch. When the program is running, which instruction fetch component performs instruction fetch and which performs instruction prefetch is determined by the decoding results of the branch type instruction, i.e., if IF_0 is responsible for instruction fetch, IF_1 is responsible for instruction prefetch, and the instruction that ends decoding is not a branch type instruction or is a branch type instruction that fails to transfer, then the two components do not change their operation, or otherwise IF_1 is responsible for instruction fetch, IF_0 is responsible for instruction prefetch; if IF_1 is responsible for instruction fetch, IF_0 is responsible for instruction prefetch, and the instruction that ends decoding is not a branch type instruction or is a branch type instruction that fails to transfer, then the two components do not change their operation, or otherwise IF_0 is responsible for instruction fetch, IF_1 is responsible for instruction prefetch.

The instruction decoding and execution module is mainly responsible for the instruction decoding and execution. The instruction that is decoded and executed is provided by an instruction fetch component in the instruction fetch module, wherein the decoding results of the branch type instruction are sent to the selector and the two instruction fetch components while those of the non-branch type instruction are sent to the instruction execution component for execution. The selector is responsible to select the instructions provided by one instruction fetch component according to the decoding results of the branch type instruction to be decoded and executed. The selector selects the instructions in IF_0 for decoding and execution when the program begins running. And in the process of program running, the selection may be done based on the decoding results of the branch type instruction. If the instructions in IF_0 are being used, and the instruction that ends decoding is not a branch type instruction or is a branch type instruction that fails to transfer, then continue using the instructions in IF_0, or otherwise, use the instructions in IF_1; if the instructions in IF_1 are being used, and the instruction that ends decoding is not a branch type instruction or is a branch type instruction that fails to transfer, then continue using the instructions in IF_1, or otherwise, use the instructions in IF_0. During instruction execution, if procedure call occurs, then the procedure return address is stored in the stack top location so that it may be prefetched.

Now refer to Fig 3. Consider that the pipeline is divided into five stages: instruction fetch (IF), instruction decoding (ID), execution (EX), memory access (MEM) and write back (WB), wherein the instruction p is the first instruction that is executed after the instruction i is executed, the instruction p+1 is the second instruction that is executed after the instruction i is executed, and the rest may be deduced by analogy. When the instruction i is decoded, which is not branch type, the instruction fetch component will read instruction p at the same time, and therefore, the ID stage of the instruction i coincides with the IF stage of the instruction p, and the EX stage of the instruction i coincides with the ID stage of the instruction p, causing no control related delay.

Now refer to Fig 4. The instruction i is branch-type. In the same pipeline as in Fig 3, the address of the instruction p can not be determined until the decoding of the instruction i is finished. As the EX stage of the instruction i coincides with the IF stage of the instruction p, there is a control related delay of a clock cycle.

Referring to Fig 5, consider the instruction I is branch type, the instruction i_1 is the instruction executed when the branch transfer fails, the instruction i_2 is the instruction executed when the branch transfer succeeds, the instruction q is the second instruction that is executed after the branch type instruction, the instruction q+1 is the third instruction that is executed after the branch type instruction, and the rest may be deduced by analogy. After the present invention is adopted, the two instruction fetch components will simultaneously read the instruction i_1 and i_2, respectively, when the instruction i is being decoded, and selects the instruction i_1 or i_2 for decoding after the decoding of the instruction i is finished. The EX stage of the instruction i coincides with the ID stage of the instruction i_1 or i_2, thereby eliminating the existent control related delay.

The present invention has already successfully realized in the pipeline of the IP core of Yinhe TS-1 embedded microprocessor and can effectively eliminate the pipeline control related delay. Fig 6 illustrates the testing results with SPECint95 benchmark set. The longitudinal axis is shown to indicate each benchmark in the SPECint95 benchmark set and the horizontal axis is shown to indicate the elimination rate of the pipeline control related delay, with 88.80% for gcc, 83.32% for ijpeg, 88.55% for compress, 88.69% for perl, 92.99% for m88ksim, 85.09% for li, 87.47% for vertex, and 86.16% for go. The average elimination rate of the pipeline control related delay is up to 87.8%.

Fig 7 illustrates the performance that are obtained with a different method for eliminating the control related delay, wherein CPI (cycle per instruction) indicates the average number of clock cycles needed in executing an instruction, and the conditional branch delay, unconditional branch delay and average branch delay are all counted in clock cycles, and wherein pipeline pause indicates no branch delay eliminating technique is applied, and double fetch components indicate that the present invention is adopted. After adopting the present invention, the conditional branch delay is reduced to 0.05, the unconditional branch delay 0.09, the average branch delay 0.06, and the effective CPI value 1.01, all of which are much lower than obtained with other methods.

Claims

What is claimed is:

1. A pipeline control related delay eliminating method, wherein the general logic structure includes a compiling module, a instruction fetch module, and an instruction decoding and execution module, wherein a compiler in the compiling module is responsible to determine all the possible transferring addresses for the branch type instruction and insert a prefetch instruction according to the category of the branch type instruction; the instruction fetch module, responsible to, in addition to providing instructions for the instruction decoding and execution module, prefetch instruction, functions through two instruction fetch components IF_0 and IF_1; a selector in the instruction decoding and execution module responsible to select the instructions provided by one of the two instruction fetch components according to the decoding results of the present branch type instruction and deliver the selected instructions to a decoding component for decoding and an instruction executing component for execution; the whole executive process being:
   1) a compiler in the compiling module determines all the possible transferring addresses for the branch type instruction and inserts a prefetch instruction;
   2) when the program begins to run, one instruction fetch component in the instruction fetch module reads in the first basic block, while the other instruction fetch component stays idle; for each basic block in the program:
      a) the instruction fetch component that is responsible to read in this basic block sequentially reads in each instruction in the basic block and determines whether it is a prefetch instruction; if the instruction that is being currently read in is a prefetch instruction, then the prefetch instruction is sent to the other instruction fetch component for it to execute the prefetch instruction, or otherwise, the instruction is sent to a decoding component for decoding;
      b) after the last instruction in the basic block, i.e., the branch type instruction, finishes decoding, a selector in the instruction decoding and execution module selects a basic block in the instruction component as the subsequent basic block of the present basic block according to the decoding results of the branch type instruction;
         i. if the instructions in IF_0 are being executed, then IF_1 executes the prefetch instruction to prefetch the subsequent target instruction for the present basic block; when instructions are sequentially executed, that is, the instruction that ends decoding is not branch type or is a branch type instruction whose transferring condition is False, the instructions in IF_0 are selected for decoding; when a branch type instruction is executed, that is, the instruction that ends decoding is a branch type instruction whose transferring condition is True, the instructions in IF_1 are selected for decoding;
         ii. if the instructions in IF_0 are being executed, then the selection

policy is just the opposite, that is, if instructions are sequentially executed, that is, the instruction that ends decoding is not branch type or is a branch type instruction whose transferring condition is False, the instructions in IF_1 are selected for decoding; when a branch type instruction is executed, that is, the instruction that ends decoding is a branch type instruction whose transferring condition is True, the instructions in IF_0 are selected for decoding;

2. A pipeline control related delay eliminating method according to Claim 1, wherein the flow of the compiler inserting a prefetch instruction is:

the compiling program sequentially reads each instruction in the program code; if a branch type instruction is read, which indicates that the end of the present basic block is reached, the compiling program inserts a corresponding prefetch instruction behind the first instruction of the basic block to which the branch type instruction belongs according to the category of the branch type instruction.

3. A pipeline control related delay eliminating method according to Claim 1, wherein the prefetch instruction is designed corresponding to various branch type instructions, including fetch addr1 and addr2, fetch addr, and fetch stack, and different prefetch instruction is applied to different branch type instruction:

1) conditional branch instruction: the transfer may succeed or fail and there are two subsequent basic blocks, and therefore, the two subsequent basic blocks of the present basic block should be prefetched simultaneously; there are two possible transferring addresses, with one address stored in the instruction and the other being the address of the following instruction; now the prefetch instruction fetch addr1 and addr2 is inserted behind the first instruction of the basic block to which the branch type instruction belongs, to prefetch the two basic blocks that begin at the addresses addr1 and addr2, respectively; addr1 is obtained by decoding the branch type instruction, and addr2 is the address of the following instruction with a value of the sum of the address of the branch type instruction and its length;
2) direct unconditional branch type instruction: the transfer is always successful and there is only one subsequent basic block, and therefore the transfer target address can be obtained and it is only required to prefetch the target basic block; there is only one possible transfer address stored in the instruction; now the prefetch instruction fetch addr is inserted behind the first instruction of the basic block to which the branch type instruction belongs, to prefetch the basic block that begins from the addresses addr; addr is obtained by decoding the branch type instruction;
3) indirect unconditional branch type instruction: the transfer is always successful, but the transfer target address is usually unobtainable for compiling as it is stored in a register. Therefore, this instruction is not processed.
4) Procedure return statement: the transfer is always successful and there is only

one subsequent basic block. This statement often appears at the procedure call return. A stack is adopted to store the return address, i.e., the prefetch address, for the procedure call as nesting may occur. At each procedure call, the return address is stored at the stack top location, from where it is obtained in prefetching. At this point, the prefetch instruction fetch stack is inserted behind the first instruction of the basic block to which the procedure return instruction belongs to prefetch the basic block that begins at the stack top location.

4. A pipeline control related delay eliminating method according to Claim 1, wherein the instruction prefetch is performed in basic blocks and all prefetched instructions will be executed except some of the subsequent instructions that are prefetched when the conditional branch instruction transfer fails.

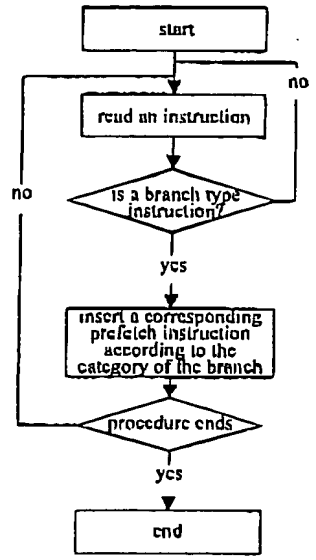


fig. 1

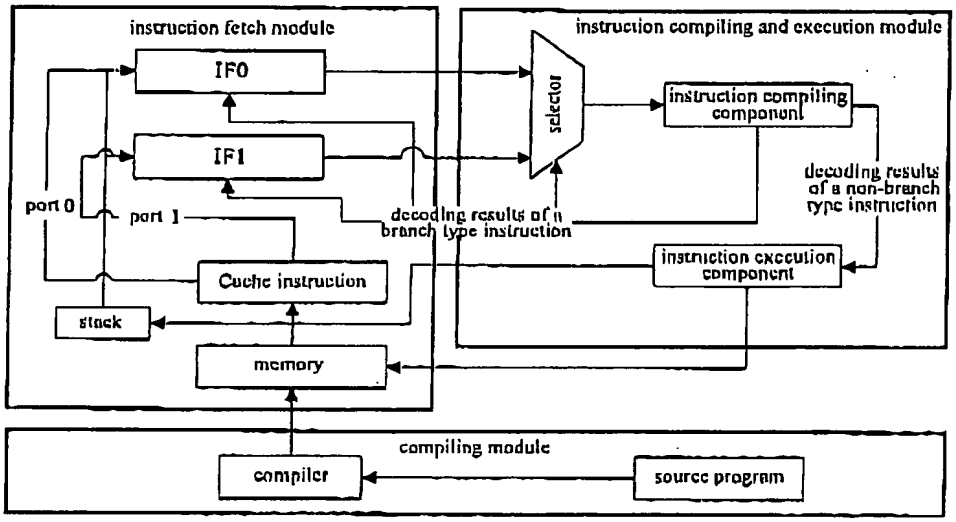


fig. 2

instruction *i*	IF	ID	EX	MEM	WB				
instruction p		IF	ID	EX	MEM	WB			
instruction p + 1			IF	ID	EX	MEM	WB		
instruction p + 2				IF	ID	EX	MEM	WB	
instruction p + 3					IF	ID	EX	MEM	WB

fig. 3

instruction i	IF	ID	EX	MEM	WB					
pause										
instruction p			IF	ID	EX	MEM	WB			
instruction p + 1				IF	ID	EX	MEM	WB		
instruction p + 2					IF	ID	EX	MEM	WB	
instruction p + 3						IF	ID	EX	MEM	WB

fig. 4

instruction i	IF	ID	EX	MEM	WB					
instruction i_1 instruction i_2		IF IF	ID	EX	MEM	WB				
instruction q				IF	ID	EX	MEM	WB		
instruction q + 1					IF	ID	EX	MEM	WB	
instruction q + 2						IF	ID	EX	MEM	WB

fig. 5

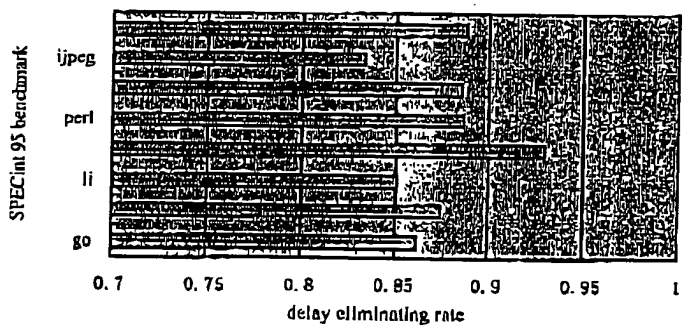


fig. 6

calling policy	conditional branch delay	unconditional branch delay	average branch delay	valid CPI
pipeline pause	1.00	1.00	1.00	1.17
transfer prediction succeeds	1.00	1.00	1.00	1.17
transfer prediction fails	0.62	0.70	0.69	1.12
delayed branch	0.25	0.00	0.21	1.04
double instruction fetch components	0.05	0.09	0.06	1.01

fig. 7

[19]中华人民共和国国家知识产权局

[51]Int. Cl[7]
G06F 9/38

[12]发明专利申请公开说明书

[21]申请号 01131569.5

[43]公开日 2002年5月15日

[11]公开号 CN 1349160A

[22]申请日 2001.11.28 [21]申请号 01131569.5

[71]申请人 中国人民解放军国防科学技术大学
地址 410073 湖南省长沙市砚瓦池正街47号中国人民解放军国防科学技术大学计算机学院
[72]发明人 戴 葵 王志英 沈 立 王蓉晖 王 雷 张春元 王明仕 刘 芳

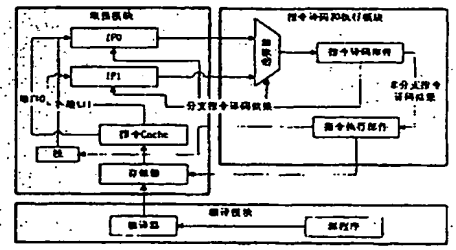
[74]专利代理机构 湖南兆弘专利事务所
代理人 赵 洪

权利要求书3页 说明书10页 附图页数2页

[54]发明名称 流水线控制相关延迟消除方法

[57]摘要

本发明公开了一种流水线控制相关延迟消除方法，其目的是在满足硬件实现简单、功耗低的前提下，高效消除流水线控制相关延迟，提高微处理器性能。技术方案是：由编译器确定分支指令所有可能的转移目标地址并插入预取指令；由两个取指令部件提前读入当前分支指令的所有后继指令，并由选择器根据对当前分支指令的译码结果选择一个取指令部件提供的指令进行译码和执行。预取指令有fetch addr1，addr2、fetch addr、fetch stack三条，它们对应不同的分支指令，由取指令部件执行。指令预取以基本块为单位。本发明硬件实现复杂度低、功耗低，控制相关延迟消除率高，无效预取指令数少。采用本发明设计的微处理器有极高性能价格比。



ISSN1008-4274

知识产权出版社出版

01.12.03

权 利 要 求 书

1．一种流水线控制相关延迟消除方法，其总体逻辑结构包括编译模块、取指令模块、指令译码和执行模块，其特征在于由编译模块中的编译器负责确定分支指令所有可能的转移地址并根据分支指令的类型插入预取指令；取指令模块除了负责为指令译码和执行模块提供指令，还要进行指令预取，其功能通过两个取指令部件 IF0 和 IF1 完成；指令译码和执行模块中有一个选择器，负责根据当前分支指令的译码结果选择一个取指令部件提供的指令交指令译码部件和指令执行部件进行译码和执行；其执行过程是：

1) 由编译模块中编译器确定分支指令所有可能的转移目标地址并插入预取指令；

2) 当程序开始运行时，取指令模块中一个取指令部件读入第一个基本块，另一个取指令部件空闲；对于程序中的每一个基本块：

(a) 负责读入该基本块的取指令部件依次读入基本块中的每一条指令，并判断它是否为预取指令；如果当前被读入的指令为预取指令，则将预取指令发送给另一个取指令部件，由它执行该预取指令；否则将指令发送给译码部件进行译码；

(b) 当基本块的最后一条指令即分支指令译码结束后，指令译码和执行模块中的选择器根据分支指令译码结果选择一个取指令部件中的基本块作为当前基本块的后继基本块：

I. 如果当前正在执行 IF_0 中的指令，IF_1 则执行预取指令，预取当前基本块的后继目标指令；当指令顺序执行，即结束译码的指令不是分支指令或转移条件为 False 的分支指令时，选

择 IF_0 中的指令进行译码；当执行分支指令时，即结束译码的指令是转移条件为 True 的分支指令时，选择 IF_1 中的指令进行译码；

II. 如果当前流水线正在执行 IF_1 中的指令，选择策略正好相反，即：如果指令依照程序顺序执行，即结束译码的指令不是分支指令或转移条件为 False 的分支指令时，选择 IF_1 中的指令进行译码；当执行分支指令时，即结束译码的指令是转移条件为 True 的分支指令时，选择 IF_0 中的指令进行译码。

2. 根据权利要求 1 所述的流水线控制相关延迟消除方法，其特征在于所述编译器插入预取指令的流程是：编译程序依次读出程序代码中的每一条指令，当遇到分支指令时，表示到达当前基本块的末尾，根据分支指令的类型在该分支指令所在基本块的第一条指令之后插入相应的预取指令。

3. 根据权利要求 1 所述的流水线控制相关延迟消除方法，其特征在于所述预取指令是对应不同的分支指令设计的，它们包括 fetch addr1, addr2、fetch addr、fetch stack 三种，不同的分支指令要使用不同的预取指令：

(1) 条件分支指令：转移可能成功也可能失败，有两个后继基本块，应同时预取当前基本块的两个后继基本块；有两个可能的转移地址，一个保存在指令中，另一个是该指令之后那条指令的地址；此时在该分支指令所在基本块的第一条指令之后插入预取指令 fetch addr1, addr2，预取从地址 addr1 和 addr2 开始的两个基本块；addr1 由该分支指令译码得到，

addr2 是该指令之后的指令地址，其值为分支指令地址加指令长度；

(2) 直接无条件分支指令：转移总是成功，只有一个后继基本块，在编译时可以获得转移目标地址，预取目标基本块即可；只有一个可能的转移地址，保存在指令中；此时在该分支指令所在基本块的第一条指令之后插入预取指令 fetch addr，预取从地址 addr 开始的基本块。addr 由该分支指令译码得到；

(3) 间接无条件分支指令：转移总是成功，但由于转移目标地址保存在寄存器中，通常在编译时无法得到，对于这类分支指令，不进行处理；

(4) 过程返回语句：转移总是成功，只有一个后继基本块，这类语句通常出现在过程调用返回时，由于过程调用可能出现嵌套，本发明用一个栈来保存过程调用的返回地址（即预取地址），每次过程调用时将返回地址保存在栈顶单元，预取时从栈顶单元中获得预取地址；此时在该过程返回指令所在基本块的第一条指令之后插入预取指令 fetch stack，预取从栈顶单元地址开始的基本块。

4. 根据权利要求 1 所述的流水线控制相关延迟的消除方法，其特征在于所述指令预取以基本块为单位进行，除了条件分支指令转移失败时预取的一部分后继指令外，所预取的指令都将被执行。

01·12·03

说　明　书

流水线控制相关延迟消除方法

技术领域：本发明涉及微处理器设计中流水线控制相关延迟的消除方法，尤其是要求功耗低、硬件实现简单的嵌入式微处理器设计中流水线控制相关延迟的消除方法。

背景技术：目前，微处理器设计中消除流水线控制相关延迟的方法大致可以分为两类，即分支预测和延迟分支方法。分支预测方法主要利用程序运行时的局部性原理，根据分支指令执行结果的统计信息预测下次分支转移是否成功。分支预测的效果不仅取决于其准确性，而且与分支预测时的开销密切相关。流水线控制相关延迟取决于流水线的结构、预测的方法和预测错误后恢复所采取的策略。这种方法的不足之处在于，预测需要大量的硬件支持，例如预测表和预测错误后的恢复部件等，实现开销大、功耗高。延迟分支方法的主要思想是在控制相关暂停周期内执行与分支指令无关的指令，从而掩盖该暂停周期。这种方法的不足之处在于，无法填充所有的分支延迟槽，同时无法保证被调度的指令总是必须要执行的，如果不是必须的，那么性能得不到真正提高。

嵌入式微处理器主要应用于家电、手机、微控制器等领域，要求功耗低，以上方法要么无法满足其功耗低、复杂度低的要求，要么控制相关延迟消除率低，无法充分提高其性能。即使在通用微处理器设计中，这些方法也无法高效率地消除流水线中的控制相关延迟。

发明内容：本发明所要解决的技术问题是在满足嵌入式微处理器

硬件实现简单、功耗低的前提下，高效消除流水线控制相关延迟，提高微处理器性能。

本发明的技术方案是：由编译模块中的编译器确定分支指令所有可能的转移目标地址并插入预取指令；在取指令模块中设计两个取指令部件提前读入当前分支指令的所有可能的后继指令；在指令译码和执行模块中设计一个选择器，由选择器根据对当前分支指令的译码结果选择一个取指令部件提供的指令交指令译码部件和指令执行部件进行译码和执行，从而消除流水线控制相关延迟。目前国内外尚无采用这种方法进行流水线控制相关延迟消除的报道。

本发明涉及到八个名词：流水线、分支指令、流水线相关、控制相关、控制相关延迟、指令预取、预取指令、基本块，它们的定义是：

(1)流水线：将微处理器的指令执行过程分解成若干个子过程，每个子过程都可以有效地在其专用功能段上与其他子过程同时执行，这就是指令执行的流水技术。流水线是流水技术的具体实现。在不同的微处理器设计中，流水线具体被分为多少个阶段也不相同。一般流水线都包括 5 个阶段：取指、译码、执行、访存、写回。

(2)分支指令：改变程序计数器值的所有指令统称为分支指令，它包括四类：条件转移指令、直接无条件转移指令、间接无条件转移指令和函数、过程返回指令（例如 return 语句）。

(3)流水线相关：由于指令间的相互依赖关系造成的流水线停顿称为流水线相关。

(4)控制相关：由于分支指令引起的流水线相关叫控制相关。

(5)控制相关延迟：因为控制相关引起的流水线暂停时钟周期数就是控制相关延迟。

(6)指令预取：在指令被执行之前提前将指令从存储器中取出的操作称为指令预取。

(7)预取指令：负责完成预取的指令称为预取指令。

(8)基本块：程序的基本组成单元，它只有一个入口（即基本块的第一条语句）和一个出口（即基本块的最后一条语句），一个基本块至少含有两条指令，基本块的出口为分支指令，一个程序总是可以划分成若干个基本块。

本发明的执行过程是：

1. 由编译模块中的编译器确定分支指令所有可能的转移目标地址并插入预取指令；

2. 当程序开始运行时，取指令模块中，一个取指令部件读入第一个基本块，另一个取指令部件空闲。

对于程序中的每一个基本块：

(a) 负责读入该基本块的取指令部件依次读入基本块中的每一条指令，并判断它是否为预取指令。如果当前被读入的指令为预取指令，则将预取指令发送给另一个取指令部件，由它执行该预取指令；否则将指令发送给指令译码和执行模块中的译码部件进行译码。

(b) 当基本块的最后一条指令即分支指令译码结束后，指令译码和执行模块中的选择器根据译码结果选择一个取指令部件中的基

本块作为当前基本块的后继基本块：

1) 假设两个取指令部件分别为 IF_0 和 IF_1，当前正在执行 IF_0 中的指令，IF_1 则执行预取指令，预取当前基本块的后继目标指令。当指令顺序执行，即结束译码的指令不是分支指令或转移条件为 False 的分支指令时，选择 IF_0 中的指令进行译码；当执行分支指令时，即结束译码的指令是转移条件为 True 的分支指令时，选择 IF_1 中的指令进行译码。

2) 如果当前流水线正在执行 IF_1 中的指令，选择策略正好相反，即：如果指令依照程序顺序执行，即结束译码的指令不是分支指令或转移条件为 False 的分支指令时，选择 IF_1 中的指令进行译码；当执行分支指令时，即结束译码的指令是转移条件为 True 的分支指令时，选择 IF_0 中的指令进行译码。

因此，在程序运行过程中，一个取指令部件负责为指令执行部件提供指令，另一个取指令部件负责完成指令预取，两个取指令部件并行工作，使得当分支指令译码结束时，所有可能的分支目标指令已经分别保存在两个取指令部件中。

与一般编译器相比，本发明的编译器增加了两项特殊功能：确定分支指令所有可能的转移地址并根据不同类型的分支指令向基本块中插入预取指令。编译器插入预取指令的流程是：编译程序依次读出程序代码中的每一条指令，当遇到分支指令时，表示到达当前基本块的末尾，根据分支指令的类型在该分支指令所在基本块的第一条指令之后插入相应的预取指令。

分支指令分为四类，本发明根据它们的不同情况设计了三条预取

指令对应不同的分支指令。三条预取指令是 fetch addr1, addr2、fetch addr、fetch stack。

(1) 条件分支指令：转移可能成功也可能失败，有两个后继基本块，应同时预取当前基本块的两个后继基本块。有两个可能的转移地址，一个保存在指令中，另一个是该指令之后那条指令的地址。此时在该分支指令所在基本块的第一条指令之后插入预取指令 fetch addr1, addr2，预取从地址 addr1 和 addr2 开始的两个基本块。addr1 由该分支指令译码得到，addr2 是该指令之后的指令地址，其值为分支指令地址加指令长度（单位为字节）。

(2) 直接无条件分支指令：转移总是成功，只有一个后继基本块，在编译时可以获得转移目标地址，预取目标基本块即可。只有一个可能的转移地址，保存在指令中。此时在该分支指令所在基本块的第一条指令之后插入预取指令 fetch addr，预取从地址 addr 开始的基本块。addr 由该分支指令译码得到。

(3) 间接无条件分支指令：转移总是成功，但由于转移目标地址保存在寄存器中，通常在编译时无法得到，对于这类分支指令，不进行处理。

(4) 过程返回语句：转移总是成功，只有一个后继基本块，这类语句通常出现在过程调用返回时，由于过程调用可能出现嵌套，本发明用一个栈来保存过程调用的返回地址（即预取地址），每次过程调用时将返回地址保存在栈顶单元，预取时从栈顶单元中获得预取地址。此时在该过程返回指令所在基本块

的第一条指令之后插入预取指令 fetch stack，预取从栈顶单元地址开始的基本块。

与其他指令不同，预取指令由取指令部件执行。不同的 RISC（精简指令集计算）指令集对应的预取指令的编码可能有所不同，但只要实现本发明相同的功能，都属于本发明保护范围。预取以基本块为单位进行，除了当条件分支指令转移失败时预取的一部分后继指令外，所预取的指令都将被执行。

如果在分支指令译码结束前所有可能的目标指令已被读入，就可以根据分支指令译码结果选择正确的后继指令进行译码和执行。采用系统性能评测协会（System Performance Evaluation Cooperative Consortium）提供的 SPECint95 基准程序组进行测试，在 FastDLX 模拟器（标准的 CPU 模拟器）中实现本发明时，如果不考虑间接无条件分支指令（这类指令在程序中所占的比率比较低），在分支指令译码结束后分支目标指令已被读入的机率达 99.3%；如果考虑间接无条件分支指令，在分支指令译码结束后分支目标指令已被读入的机率达 93%。

本发明具有以下优点：

(1) 硬件实现复杂度低、功耗低。与分支预测技术相比，本发明省去了复杂的分支预测硬件和预测错误时的恢复硬件，仅仅使用了简单的控制逻辑部件，大大降低了硬件实现的难度和复杂度。

(2) 控制相关延迟消除率高。本发明直接根据分支指令的译码结果选择分支目标指令，指令预取保证了当分支指令译码结束

后绝大多数指令都已被读入，从而消除了绝大多数控制相关延迟。

(3) 预取以基本块为单位进行，在执行当前基本块的同时预取它的所有后继基本块，使得预取操作发出时间早，保证了有充足时间完成预取；同时，除了条件分支指令转移失败时预取的一部分后继指令外，所预取的指令都是将要被执行的，有效地减少了无效预取数。

本发明在满足嵌入式微处理器硬件实现简单、功耗低的前提下，实现了高效消除流水线控制相关延迟，提高微处理器性能的目的。本发明也可应用在通用微处理器设计中。

附图说明：

图 1 是本发明编译器插入预取指令的流程图；

图 2 是本发明总体逻辑结构图；

图 3 是一般微处理器非分支指令在 5 级流水线中的时空图；

图 4 是一般微处理器分支指令在 5 级流水线中的时空图；

图 5 是采用本发明后分支指令在 5 级流水线中的时空图；

图 6 是采用本发明针对 SPECint95 基准程序的测试结果；

图 7 是采用本发明和其它控制相关延迟消除方法的性能比较。

具体实施方式：

图 1 为本发明编译器插入预取指令的流程图。编译程序依次读出程序代码中的每一条指令，当遇到分支指令时，表示到达当前基本块的末尾，根据分支指令的类型在该分支指令所在基本块的第一条指令之后插入相应的预取指令，程序最后总是一条过程返回语句，因此编

译时总能遇到分支指令。

图 2 是本发明的总体逻辑结构图。它由编译模块、取指令模块、指令译码和执行模块组成：

编译模块主要负责确定分支指令所有可能的转移地址并根据分支指令的类型插入预取指令。编译器依次读取源程序中的每一条指令，如果该指令为分支指令，则根据分支指令的类型向它所在基本块的第一条指令之后插入相应的预取指令。具体方法为：对于条件分支指令，有两个后继基本块，相应的预取地址有两个，插入预取指令 fetch addr1，addr2，其中 addr1 由该分支指令译码得到，addr2 是该指令之后的指令地址，其值为分支指令地址加指令长度（单位为字节）；对于直接无条件分支指令，只有一个后继基本块，相应的预取地址有一个，插入预取指令 fetch addr，其中 addr 由指令译码得到；过程返回语句，有一个后继基本块，相应的预取地址有一个，保存在栈顶单元中，插入预取指令 fetch stack。编译后的程序代码保存在存储器中。

取指令模块主要负责为指令译码和执行模块提供指令，并进行指令预取，其功能通过两个取指令部件完成。取指令部件 IF_0 通过端口 0 从指令 Cache 中读指令，取指令部件 IF_1 则通过端口 1 从指令 Cache 中读指令。在程序开始运行时，选择 IF_0 提供的指令进行译码和执行，IF_1 空闲，当 IF_0 读入预取指令后，将其转发给 IF_1，由 IF_1 完成预取；在程序运行过程中，哪个取指令部件进行取指令，哪个取指令部件进行指令预取应根据分支指令的译码结果确定：如果 IF_0 负责取指令，IF_1 负责指令预取，结束译码的指令不是分支指令或者是转移失败的分支指令，那么二者操作不变，否则 IF_1 负责取指令，IF_0 负责指令预

取；如果 IF_1 负责取指令，IF_0 负责指令预取，结束译码的指令不是分支指令或者是转移失败的分支指令，那么二者操作不变，否则 IF_0 负责取指令，IF_1 负责指令预取。

指令译码和执行模块主要负责指令的译码和执行，被译码和执行的指令由取指令模块中的一个取指令部件提供，其中分支指令的译码结果被送往选择器和两个取指令部件，非分支指令的译码结果被送往指令执行部件进行执行。选择器负责根据分支指令的译码结果选择一个取指令部件提供的指令进行译码和执行，当程序开始运行时，它选择 IF_0 中的指令译码和执行，在程序运行过程中，它根据分支指令的译码结果进行选择：如果正在使用 IF_0 中的指令，结束译码的指令不是分支指令或者是转移失败的分支指令，那么继续使用中 IF_0 的指令，否则使用 IF_1 中的指令；如果正在使用 IF_1 中的指令，结束译码的指令不是分支指令或者是转移失败的分支指令，那么继续使用 IF_1 中的指令，否则使用 IF_0 中的指令。在指令执行过程中，如果发生过程调用，则将过程返回地址保存在栈顶单元中以便进行预取。

如图 3，假设流水线分为 5 个阶段：取指（IF）、译码（ID）、执行（EX）、访存（MEM）和写回（WB），指令 p 为在指令 i 之后执行的第一条指令，指令 p+1 为在指令 i 之后执行的第二条指令，依次类推。由于 i 不是分支指令，那么在对它译码的同时，取指令部件将读取指令 p，因此指令 i 的 ID 阶段与指令 p 的 IF 阶段重合，指令 i 的 EX 阶段与指令 p 的 ID 阶段重合，此时没有控制相关延迟。

如图 4，指令 i 为分支指令，在与图 3 相同的流水线中，只有在指令 i 译码结束后，才能确定指令 p 的地址，指令 i 的 EX 阶段与指令 p

的 IF 阶段重叠，此时流水线有一个时钟周期的控制相关延迟。

如图 5，假设指令 I 为分支指令，指令 i_1 是分支转移失败时执行的指令，指令 i_2 是分支转移成功时执行的指令，指令 q 为在分支指令后执行的第二条指令，指令 q+1 为在分支指令后执行的第三条指令，依此类推。采用本发明后，在指令 i 译码的同时，两个取指令部件将分别读取指令 i_1 和 i_2，在指令 i 译码结束后即可根据译码结果选择指令 i_1 或 i_2 进行译码，指令 i 的 EX 阶段与指令 i_1 或 i_2 的 ID 阶段重合，消除了原有的控制相关延迟。

本发明已成功地在银河 TS－1 嵌入式微处理器 IP 核的流水线中实现，能有效消除流水线控制相关延迟，图 6 为采用 SPECint95 基准程序组测试得到的结果。图中，纵轴表示 SPECint95 基准程序组中的每个基准程序，横轴表示控制相关延迟消除率，gcc 为 88.80%，ijpeg 为 83.32%，compress 为 88.55%，perl 为 88.69%，m88ksim 为 92.99%，li 为 85.09%，vertex 为 87.47%，go 为 86.16%，流水线控制相关延迟的平均消除率达到 87.8%。

图 7 出了采用不同的控制相关延迟消除方法所得到的性能，其中 CPI（cycle per instruction）表示执行一条指令所需的平均时钟周期数，条件分支延迟、无条件分支延迟和平均分支延迟的单位都是时钟周期（cycle）。其中，流水线暂停代表没有采用任何分支延迟消除技术的情况，双取指部件表示采用本发明的情况。采用本发明后，条件分支延迟降低到 0.05，无条件分支延迟降低到 0.09，平均分支延迟降低到 0.06，有效 CPI 值降低到 1.01，均远远低于采用其他方法得到的结果。

说 明 书 附 图

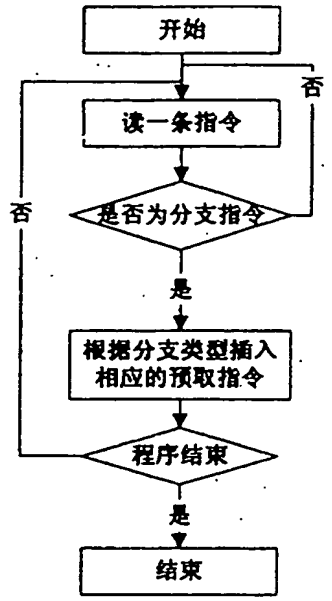


图 1

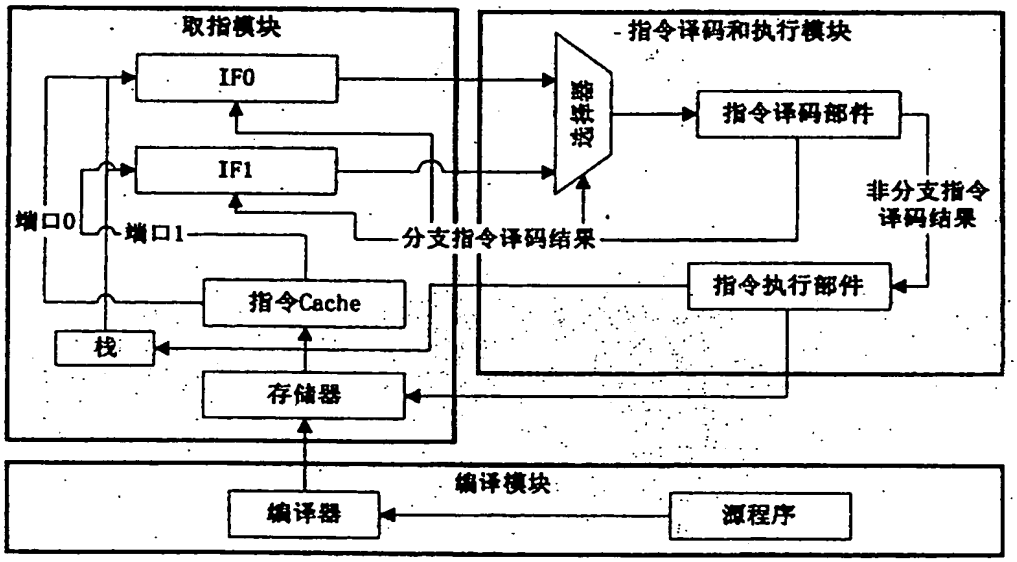


图 2

指令 i	IF	ID	EX	MEM	WB				
指令 p		IF	ID	EX	MEM	WB			
指令 p+1			IF	ID	EX	MEM	WB		
指令 p+2				IF	ID	EX	MEM	WB	
指令 p+3					IF	ID	EX	MEM	WB

图 3

指令 i	IF	ID	EX	MEM	WB					
暂停										
指令 p			IF	ID	EX	MEM	WB			
指令 p + 1				IF	ID	EX	MEM	WB		
指令 p + 2					IF	ID	EX	MEM	WB	
指令 p + 3						IF	ID	EX	MEM	WB

图 4

指令 i	IF	ID	EX	MEM	WB					
指令 i_1 指令 i_2		IF IF	ID	EX	MEM	WB				
指令 q				IF	ID	EX	MEM	WB		
指令 q + 1					IF	ID	EX	MEM	WB	
指令 q + 2						IF	ID	EX	MEM	WB

图 5

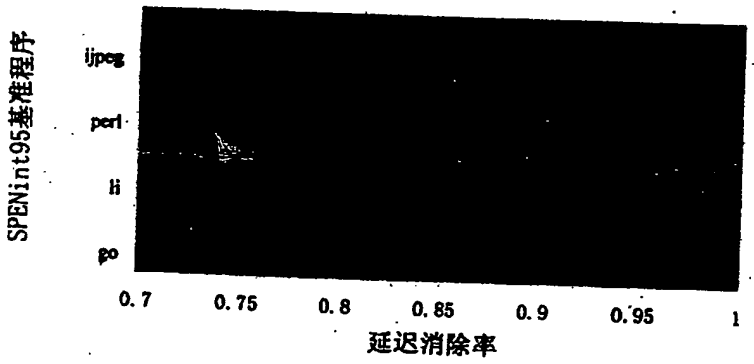


图 6

调度策略	条件分支延迟	无条件分支延迟	平均分支延迟	有效 CPI
暂停流水线	1.00	1.00	1.00	1.17
预测转移成功	1.00	1.00	1.00	1.17
预测转移不成功	0.62	0.70	0.69	1.12
延迟分支	0.25	0.00	0.21	1.04
双取指部件	0.05	0.09	0.06	1.01

图 7

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☐ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☑ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.